Our Ref.: 042390.P13415

APPLICATION FOR UNITED STATES LETTERS PATENT

FOR

Method and System to Use and Maintain a Return Buffer

Inventors: **John Alan Miller**
Michael J. St. Clair

Prepared by:

BLAKELY SOKOLOFF TAYLOR & ZAFMAN, LLP
12400 Wilshire Boulevard, 7th Floor
Los Angeles, California 90025
(503) 684-6200

Method and System to Use and Maintain a Return Buffer

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to the field of return buffers. In particular, the invention relates to using and maintaining a return buffer in an instruction pipeline of a microprocessor.

2. Description of Related Art

Microprocessors have instruction pipelines in order to increase program execution speeds. However, when there is a branch instruction, the pipeline may stall while waiting for the branch condition to resolve before fetching the next instruction, since it is not known whether the branch will be taken. In order to prevent stalls in the pipeline while waiting for the next instruction to be fetched, microprocessors employ branch prediction circuitry to predict whether a branch will be taken. Certain types of branch instructions are associated with program calls to subroutines. A typical program calls a subroutine by issuing a call instruction, citing the address of the subroutine to which the program should branch. The subroutine typically ends with a return instruction, which causes a branch back to the program that issued the call. When the call instruction is executed, the address of the next sequential instruction is pushed onto a branch prediction buffer called a return buffer. When the return instruction is retired, the return buffer is popped, thereby providing the appropriate return address.

In a microprocessor that has multiple instruction sources, such as instruction fetch, instruction decode, and branch prediction, multiple pointers from each

instruction source to the return buffer are maintained and the return address is sent from the return buffer to the appropriate instruction source when needed. However, there is the problem of latency for instruction sources or pipeline units that are not located close to the return buffer. Sources or units not located close to the return buffer may take a long time to obtain a correct return address from the return buffer, thus requiring a stall in the pipeline to wait for the return address.

BRIEF DESCRIPTION OF DRAWINGS

The invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings in which like reference numerals refer to similar elements.

Figure 1 is a block diagram illustrating one embodiment of the invention.

Figure 2 illustrates one embodiment of a return buffer maintained by an instruction pipeline according to the invention.

Figure 3 illustrates one embodiment of a return buffer maintained with multiple pointers according to the invention.

Figure 4 is a flowchart that illustrates one embodiment of the process of the invention performed by an instruction pipeline upon detecting a call instruction.

Figure 5 is a flowchart that illustrates one embodiment of the process of the invention performed by an instruction pipeline upon detecting a return instruction.

DETAILED DESCRIPTION OF THE INVENTION

Embodiments of a system and method for using and maintaining a return buffer are described. In the following description, numerous specific details are provided for a thorough understanding of embodiments of the invention. One skilled in the relevant art will recognize, however, that the invention can be practiced without one or more of the specific details, or with other methods, components, materials, etc. In other instances, well-known structures, materials, or operations are not shown or described in detail to avoid obscuring aspects of the invention.

Reference throughout this specification to "one embodiment" or "an embodiment" means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment of the invention. Thus, the appearances of the phrases "in one embodiment" or "in an embodiment" in various places throughout this specification are not necessarily all referring to the same embodiment. Furthermore, the particular features, structures, or characteristics may be combined in any suitable manner in one or more embodiments.

Referring to Figure 1, one embodiment of an instruction pipeline 100 implementing the invention is illustrated. Those of ordinary skill in the art will appreciate that the instruction pipeline 100 may include more components than those shown in Figure 1. However, it is not necessary that all of these generally conventional components be shown in order to disclose an illustrative embodiment for practicing the invention. In one embodiment of the invention, instruction pipeline 100 includes a plurality of pipeline units. For example, three pipeline units 102, 104, and 106 are illustrated in Figure 1. Although three pipeline units are illustrated, the instruction pipeline 100 may include more or fewer units. Although the pipeline units of instruction pipeline 100 are illustrated in series, alternative arrangements are

possible. For example, a pipeline unit may be connected in parallel with another pipeline unit.

In one embodiment of the invention, pipeline unit 102 may be an instruction source for instruction pipeline 100. For example, pipeline unit 102 may fetch instructions from main memory or cache and provide the fetched instructions to the instruction pipeline 100 for processing. In another embodiment of the invention, pipeline unit 102 may be an instruction decode unit that decodes instructions and then passes them on to another unit for processing. In the alternative, pipeline unit 102 may be a branch prediction unit that predicts whether branch instructions will be taken. In yet another embodiment of the invention, pipeline unit 102 may be a cache branch prediction unit for predicting branches in the cache. Those with ordinary skill in the art will appreciate that each pipeline unit may be any of various types of units, including but not limited to those described above. After each pipeline unit processes the instruction, the instruction is passed along to the next unit in the pipeline.

Instruction pipeline 100 processes various types of instructions, including subroutine call instructions. As explained above, a typical program calls a subroutine by issuing a call instruction. The called subroutine typically ends with a return from subroutine instruction. Instruction pipeline 100 includes one or more pipeline units that maintain a return buffer onto which return addresses are pushed. In one embodiment, the return buffer is a return stack. In the exemplary pipeline illustrated in Figure 1, three return buffers 108, 110, and 112 are maintained by pipeline units 102, 104, and 106 respectively.

Each return buffer may be of equal or varying depth. For example, return buffer 112 may have greater depth, that is, contain more entries than return buffer 110. Each pipeline unit that maintains a return buffer maintains a pointer to that return buffer. For example, pipeline unit 102 maintains a pointer 120 to its return

buffer 108. Likewise, pipeline unit 104 maintains a pointer 122 to its return buffer 110. A pipeline unit may maintain more than one pointer for its return buffer. For example, pipeline unit 106 may maintain three pointers, 114, 116, and 118, for pipeline units 102, 104, and 106 respectively. In the alternative, a pipeline unit may maintain pointers to more than one return buffer. For example, pipeline unit 102 may maintain a pointer 120 to return buffer 108 and a pointer 114 to return buffer 112. This way, if one return buffer becomes corrupted or has an incorrect return address, there is another source available from which to obtain the correct return address.

Referring to Figure 2, one embodiment of a return buffer 108 according to the invention is illustrated. The return buffer 108 has a number of entries 202. In one embodiment of the invention, each entry in the return buffer 108 contains two fields: a return address 204 and a valid bit 206. The valid bit 206 indicates whether the return address 204 is valid. In one embodiment of the invention, the return buffer is a return stack, and a unit pointer 120 is maintained to point to the return address at the top of the stack. New entries are pushed onto the top of the stack. When a return address is needed, the entry at the top of the stack is popped. The return address may then be used in the instruction pipeline.

Referring to Figure 3, one embodiment of a return buffer 112 with multiple pointers is illustrated. Return buffer 112 has a number of entries 302. Each entry includes a return address 304. More than one unit pointer is maintained for return buffer 112. For example, three unit pointers 114, 116, and 118 are maintained for three different pipeline units. Each pointer may point to the same or different entries in return buffer 112. The different pointers allow different pipeline units to read the entries in return buffer 112 when return addresses are needed.

Referring to Figure 4, a flow chart illustrates one embodiment of the process of the invention performed by a first pipeline upon detecting a call instruction. First,

at 400, the first pipeline unit pushes the return address onto the top of its return buffer. In one embodiment of the invention, the first pipeline unit may be an instruction decode or instruction translation unit. Then, at 402, the valid bit for the return buffer entry containing the return address is set to valid. Next, at 404, the first pipeline unit updates its pointer. For example, the first pipeline unit may increment the pointer. Then, at 406, the first pipeline unit determines if a second pipeline unit predicted the call instruction. In one embodiment of the invention, the second pipeline unit is a branch prediction unit (BPU) for predicting whether branches will be taken, and the first pipeline unit determines if the BPU predicted the call instruction. If the BPU predicted the call instruction, nothing needs to be done, and the process may end. If the BPU did not predict the call instruction, then, at 408, the first pipeline unit sends the return address at the top of its return buffer (TOS address) to the BPU.

Referring to Figure 5, a flow chart illustrates one embodiment of the process of the invention performed by the first pipeline upon detecting a return instruction. First, at 500, the first pipeline unit pops its return buffer. Then, at 502, the first pipeline unit checks the valid bit of the popped entry to determine if the popped return address is valid. Next, at 504, the first pipeline unit updates its pointer. For example, the first pipeline unit may decrement the pointer. Then, at 506, the first pipeline unit sends a request to a third pipeline unit to fill the first pipeline unit's return buffer with entries from the third pipeline unit's return buffer. In one embodiment of the invention, the third pipeline unit is a branch prediction unit for predicting whether branch instructions will be taken. In one embodiment of the invention, the branch prediction unit may be a cache branch prediction unit for predicting branches in a cache or a trace branch prediction unit (TBPU) for predicting branches in a trace cache.

The third pipeline unit updates its return buffer pointers after sending its return buffer entries to the first pipeline unit. Then, at 508, the first pipeline unit determines if the second pipeline unit predicted the return instruction. If the second pipeline unit predicted the return instruction, nothing needs to be done and the process may end. If the second pipeline unit did not predict the return instruction, then, at 510, the first pipeline unit sends the return address at the top of its return buffer (TOS address) to the second pipeline unit.

In one embodiment of the invention, the instruction pipeline checks the cache to determine if there is a cache hit. The cache may be a trace cache. The instruction pipeline keeps looking in the cache until there is a hit. When there is a hit, a clear signal is sent to the decode unit, and the decode unit may clear its return buffer. The instruction pipeline will then execute instructions from the cache as long as there continues to be a cache hit. Then, when there is a cache miss, the TBPU may notify the decode unit and the decode unit then sends a request to the TBPU to fill the decode unit's return buffer with entries from the TBPU's return buffer.

An illustrative example of the method according to the invention will now be described. For purposes of illustration, assume that the instruction pipeline includes three pipeline units that each maintain a return buffer: the TBPU, BPU, and instruction translation unit (IXLAT). The return buffers in this example are return stacks. Three pointers are maintained with the TBPU return buffer: a TBPU pointer, a BPU pointer, and an IXLAT pointer. Assume that the TBPU return buffer contains the following return addresses: 0XA000, 0XB000, 0XC000, 0XD000, etc. The address 0XA000 is at the top of the stack. Initially, all three pointers are pointing to the top of the stack. The IXLAT return buffer is a two-entry deep return buffer for purposes of this example. Initially, there are no valid entries in the IXLAT return buffer. The BPU return buffer is a one-entry deep return buffer for purposes of this example.

Assume that the instruction pipeline is executing out of the cache. This continues as long as there is a cache hit. When there is a cache miss, the TBPU notifies the IXLAT. The IXLAT then sends a request to the TBPU for a return address. The TBPU sends the IXLAT the address pointed to by the IXLAT pointer: address 0XA000. The IXLAT pushes this return address onto the top of its return buffer, sets the valid bit for this entry to valid, and updates its top of stack pointer to point to address 0XA000. The TBPU also updates the IXLAT pointer to point to the next return address, which is 0XB000. Then, since the IXLAT return buffer still contains one empty entry, the IXLAT sends a request to the TBPU to fill the IXLAT return buffer. The TBPU sends the address 0XB000 to the IXLAT, and then updates the IXLAT pointer to point to the next address, which is 0XC000. The IXLAT puts the address 0XB000 into its return buffer and sets the valid bit to valid. The IXLAT return buffer is now full with two entries: 0XA000 (at the top of the stack) and 0XB000.

During decode, when the IXLAT detects a call instruction, it will push the return address onto its return buffer. For example, if the IXLAT detects a call instruction with a return address 0XF000, the IXLAT will push the address 0XF000 onto the top of its return buffer, set the valid bit to valid, and update its top of stack pointer to point to the address 0XF000. The IXLAT return buffer now has two entries: 0XF000 (at the top of the stack) and 0XA000. The IXLAT then determines if the BPU predicted the call instruction. If the BPU did not predict the call instruction, the IXLAT will send the address 0XF000 to the BPU. The BPU will then push this address onto the top of its return buffer.

If the IXLAT detects a return instruction, it will pop its return buffer and retrieve the address 0XF000. The IXLAT will check the valid bit of the popped address to determine if the address is valid. If the address is valid, it may then be used in the instruction pipeline. The IXLAT will update its top of stack pointer to

point to the next address 0XA000. Then, the IXLAT will send a request to the TBPU to fill the IXLAT return buffer. The IXLAT will determine if the BPU detected the return instruction. If the BPU did not detect the return instruction, the IXLAT will send the top of stack address 0XA000 to the BPU. If the IXLAT detects another return instruction while waiting for the TBPU to fill the IXLAT return buffer, the IXLAT will pop the address 0XA000. Then, the IXLAT return buffer will be empty. If the IXLAT detects another return instruction while its return buffer is empty, it will stall until it receives entries from the TBPU return buffer.

The above description of illustrated embodiments of the invention, including what is described in the Abstract, is not intended to be exhaustive or to limit the invention to the precise forms disclosed. While specific embodiments of, and examples for, the invention are described herein for illustrative purposes, various equivalent modifications are possible within the scope of the invention, as those skilled in the relevant art will recognize.

These modifications can be made to the invention in light of the above detailed description. The terms used in the following claims should not be construed to limit the invention to the specific embodiments disclosed in the specification and the claims. Rather, the scope of the invention is to be determined entirely by the following claims, which are to be construed in accordance with established doctrines of claim interpretation.
